j0205582**Basra University**

**Collage Of Computer Science &**

**Information Technology**

**Computer Science Department**

CS204

Microprocessor

And

Assembly Language Laboratory

العملــــــي ))

**2015 -2016**

**طاقم تعليمات المعالج الدقيق 8088 / 8086**

من الجدير بالملاحظة ان كلا من المعالج الدقيق 8088 و 8086 لهما نفس طاقم التعليمات ويمكن تقسيمها إلى سبعة مجموعات مختلفة حسب الوظيفة وكما يلي :

* 1. تعليمات نقل البيانات Data Movement Instructions
  2. التعليمات الحسابية Arithmetic Instructions
  3. تعليمات معالجة البيانات الثنائية Bit Control Transfer Instructions
  4. تعليمات نقل التحكم Control Transfer Instructions
  5. تعليمات معالجة سلاسل الرموز String Instructions
  6. تعليمات الاعتراضات Interrupt Instructions
  7. تعليمات التحكم بالمعالج Processor Control Instruction

وفي الأجزاء التالية سنستعرض بالتفصيل إلى تلك التعليمات :

بشكل عام ، ان أي تعليمة بلغة الاسمبلي تأخذ الشكل التالي :

[ label : ] [ Mnemonic ] [ operand ] [ ; comment ]

حيث ان :

Mnemonic : يمثل العملية التي سيتم تنفيذها مثل ، Add , Mov , sub , mul ,…..

Operands : يتم بواسطتها نقل البيانات واستخدامها في التعليمات وان المعلمات تأخذ أنواع مختلفة كالأتي :

**Operand types:**  
  
**REG**: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.  
  
**SREG**: DS, ES, SS, and only as second operand: CS.  
  
**memory**: [BX], [BX+SI+7], variable, etc...  
  
**immediate**: 5, -24, 3Fh, 10001101b, etc...

**1) تعليمات نقل البيانات Data Movement Instructions**

تقوم هذه التعليمات بنقل البيانات والعناوين ما بين المسجلات والذاكرة أو موانئ الإدخال والإخراج . وسنذكر أهمها كما يلي :

1. **تعليمة نقل او نسخ المعلومات Mov:** تعمل هذه التعليمة على نقل بايت او كلمة ما بين مسجل او مكان في الذاكره او ما بين مسجلين . وكذلك يمكن عن طريق هذه التعليمة نقل قيمة فورية موجودة ضمن التعليمة الى مسجل او مكان في الذاكرة والصيغة العامة لهذه التعليمة كما يلي :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mnemonic** | **Meaning** | **Format** | **Operation** | **Flag effected** |
| MOV | Move | Mov D,S | D ←S | None |

|  |  |
| --- | --- |
| **MOV** | **REG, memory memory, REG REG, REG memory, immediate REG, immediate SREG, memory memory, SREG REG, SREG SREG, REG** |

**ملاحظات :**

1. يستخدم مع جميع السجلات

2- يحتوي الإيعاز على معلمين source و destination (مصدر وهدف) تقوم بنقل او نسخ قيمة المصدر الى الهدف كما في الشكل . Mov destination , source

* أن تعليمة mov لا يمكنها القيام بالوظائف التالية :
  1. لا يمكن إسناد قيمة مباشرة إلى كل من المسجلين CS و IP .
  2. لا يمكن نسخ محتويات من مسجل مقطعي إلى مسجل مقطعي اخر بشكل مباشر ( يجب ان يتم نسح المكونات أولا إلى مسجل عام أولا ثم إلى المسجل المقطعي) .
  3. لا يمكن نسخ قيم فورية الى مقطع مسجل ( يجب أن يتم نسخ القيم أولا إلى مسجل عام )
  4. لا يمكم نقل البيانات من موقع ذاكره الى موقع ذاكرة آخر بشكل مباشر .

**أمثلة عن الحالات المسموح بها في عملية mov :**

Mov Ax , 0000 ; ax=0000 **🗸**

Mov Ax , bx ; bx=ax **🗸**

Mov cx , 01234h  **🗸**

Mov dl , 10110001b **🗸**

Mov ah , 24h **🗸**

**أمثلة عن الحالات الغير مسموح بها في تعليمة Mov**

Mov CS ,bx 🗶

Mov IP , ax 🗶

Mov CS , DS 🗶

Mov Es , 4567 🗶

**2- تعليمــــة XCHG :** تستعمل هذه التعليمة لتبديل قيمة مسجل في الذاكرة مع مسجل اخر او مع موقع في الذاكرة بنفس الطول . طول المسجلات او مواقع الذاكرة التي يمكن تبديلها مع بعضها البعض يمكن ان تكون بايت او كلمة . لا يجوز تبديل قيم مسجلات القطاعات مع بعضها البعض عن طريق هذه التعليمة . وفيما يلي الصيغة العامة لهذه التعليمة :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mnemonic** | **Meaning** | **Format** | **Operation** | **Flag effected** |
| XCHG | Exchange | XCHG D,S | D ↔S | None |

|  |  |
| --- | --- |
| XCHG | REG, memory memory, REG REG, REG |

Example:

Org 100h

MOV AL, 5

MOV AH, 2

XCHG AL, AH ; AL = 2, AH = 5

RET

**2- التعليمات الحسابية Arithmetic Instructions**

توفر لغة التجميع تعليمات حسابية للعمليات الحسابية الأساسية والتي تتكون من عمليات الجمع والطرح والضرب والقسمة وكما هو مبين أدناه :

**\* أيعازات الجمع Add instruction**

توفر لغة الاسمبلي عدد من ايعازات الجمع ، مع ملاحظة ان هذه الايعازات تطبق على الأعداد السداسية وليس الاعداد العشرية . وسنتطرق ألان الى اهم تلك تلك الايعازات :

1. **تعليمة Add :**

ان تعليمة Add تأخذ الشكل التالي Add opreand1 , operand2

حيث يتم جمع قيمة المعاملين ووضعها في المعامل الثاني **operand1 = operand1 + operand2**

والجدول التالي يبن الحالات المسوح بها في تعليمة Add

|  |  |
| --- | --- |
| **ADD** | **REG, memory memory, REG REG, REG memory, immediate REG, immediate** |

من الملاحظ أن تعليمة Add تغير في حالة سجل الحالة الذي سيأتي ذكره بالتفصيل لاحقاً .

**org 100h**

**MOV AL, 5 ; AL = 5**

**ADD AL, -3 ; AL = 2**

**RET**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mnemonic** | **Meaning** | **Format** | **Operation** | **Flag effected** |
| ADD | Addition | ADD D , S | D ← D + S , Carry → CF | أعلام الحالة |
| ADC | Add with carry | ADC D , S | D ← D + S + CF ,  Carry → CF | أعلام الحالة |
| INC | Increment by 1 | INC D |  | أعلام الحالة |

\* من الجدول السابق نلاحظ أن التعليمات الرياضية تؤثر على قيم سجلات الحالة :

**2- تعليمة الإضافة بواحد Inc (increment by 1 )**

|  |  |
| --- | --- |
| INC | REG memory |

و الصيغة العامة **inc operand1 ،** أي أن **operand1 = operand1 + 1**

حيث تستخدم هذه التعليمة لأضافه واحد إلى قيمة مسجل أو ذاكره ، وان قيم سجلات الحالة تتغير حسب نتيجة التعليمة :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Z | S | O | P | A |
| r | r | r | r | r |

CF - unchanged!

**مثال :**

org 100h

mov bx , 12

inc bx

ret

**3- تعليمة ADC (Add with carry )**

الصيغة العامة لهذه التعليمة **ADC operand1 , operand2**

|  |  |
| --- | --- |
| ADC | REG, memory memory, REG REG, REG memory, immediate REG, immediate |

حيث يتم جمع قيم المعاملين مع قيمة علم carry flag **operand1 = operand1 + operand2 + CF**

وهذه التعليمة تغير أيضا من حالة سجلات الحالة حسب نتيجة التعليمة :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| r | r | r | r | r | r |

وكما موضح في المثال التالي :

Org 100 h

STC ; set CF = 1

MOV AL, 5 ; AL = 5

ADC AL, 1 ; AL = 7

RET

**4- التعليمة STC ( set carry flag ) :**

وضيفتها أعطاء قيمة 1 إلى carry flag ، وتأخذ الصيغة التالية : **STC**

|  |  |
| --- | --- |
| STC | No operands |

وكما نلاحظ أن هذه التعليمة لا تحتاج أي معلمة ولكنها فقط تغير قيمة carry flag أي ان CF = 1

|  |
| --- |
| C |
| 1 |

**\* تعليمات الطرح Subtraction Instruction**

**1- تعليمة الطرح Sub :**

وتأخذ الصيغة العامة التالية : **sub operand 1 , opearnd2**

حيث يتم طرح المعامل الثاني من الاول والنتيجة في المعامل الأول **operand1 = operand1 - operand2**

|  |  |
| --- | --- |
| SUB | REG, memory memory, REG REG, REG memory, immediate REG, immediate |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mnemonic** | **Meaning** | **Format** | **Operation** | **Flag effected** |
| SUB | Subtract | SUB D , S | D ← D - S ,  Borrow → CF | أعلام الحالة |
| SBB | Subtract with borrow | SBB D , S | D ← D - S + CF ,  carry → CF | أعلام الحالة |
| DEC | Decrement by 1 | DEC D | D ← D - 1 | أعلام الحالة |
| NEG | Negate | NEG |  | أعلام الحالة |

يقوم المعالج بأجراء عمليات الطرح بواسطة وحدة الجمع (Adder) وذلك بجمع المطروح منه مع المكمل الثنائي للمطروح حيث ان المعالج لايحتوي على وحدة داخلية خاصة بعملية الطرح , ويمكن توضيح هذه الفقرة كما يلي : 10 + (-7) = 10 -7 ، اذا يجب ايجاد المكمل الثنائي للعدد 7 ثم يجمع هذا المكمل للمطروح منه أي العدد 10 ، وفي هذه الحالة يتم اهمال راية الحمل :

العدد 10 ممثلا ب8 خانات ثنائية 0000 1010

+

المكمل الثنائي للعدد 7 1111 1001 -

وتعطي هذه العملية النتيجة 3 = 1 0000 0011

وكما في المثال التالي :

org 100h

mov al, 10 h

mov bl , 7

sub al, bl

ret

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| r | r | r | r | r | r |

وان حالة رايات الحالة تتغير تبعاً لنتيجة تنفيذ التعليمة :

1. **تعليمة الطرح بواحد ( DEC (**Decrement by 1

تستخدم هذه التعليمة لطرح قيمة واحد من المعامل وتاخذ الصيغة التالية : DEC operand1

حيث أن : operand1 = opearnd1 – 1

|  |  |
| --- | --- |
| DEC | REG memory |

**مثال :**

org 100h

MOV AL, 255 ; AL = 0FFh (255 or -1)

DEC AL ; AL = 0FEh (254 or -2)

RET

اما اعلام الحالة فتتغير حسب نتيجة التعليمة ماعدا علم الحمل فانه لا يتغير

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Z | S | O | P | A |
| r | r | r | r | r |

CF - unchanged!

1. **تعليمة الطرح SBB(Subtract with borrow)**

وتأخذ الصيغة التالية : **SBB operand1 , operand2**

حيث ان : **operand1 = operand1 - operand2 - C**F ، أي يتم طرح قيمة المعامل الأول من الثاني وكذلك من قيمة راية الحمل اذا وجدت :

|  |  |
| --- | --- |
| SBB | REG, memory memory, REG REG, REG memory, immediate REG, immediate |

وأيضا سيتم تغيير أعلام الحالة حسب نتيجة التعليمة :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| r | r | r | r | r | r |

مثال :

org 100h

STC

mov al, 9 h

mov bl , 7

SBB al, bl

Ret

1. **تعليمة النفي NEG (Negative )**

تستخدم لقلب بتات المعامل ثم إضافة واحد وتأخذ الصيغة التالية : NEG operand1

حيث ان:

|  |  |
| --- | --- |
| NEG | REG memory |

**حيث تقوم بالعمل التالي** :

Algorithm:

* Invert all bits of the operand
* Add 1 to inverted operand

**مثال** :

Example:

Org 100h

MOV AL, 5 ; AL = 05h

NEG AL ; AL = 0FBh (-5)

NEG AL ; AL = 05h (5)

RET

وأيضا سيتم تغيير أعلام الحالة حسب نتيجة التعليمة

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| r | r | r | r | r | r |

**5- عمليات الضرب والقسمة Multiplication and Division Instructions**

يتم تطبيق هذه التعليمات على الأعداد الثنائية او بالشفرة BCD ويوجد عدة أنواع من التعليمات ضمن هذه المجموعة كما يلي :

**1- تعليمات الضرب للأعداد بدون إشارة - MUL Unsigned Numbers Multiply**

تستعمل هذه التعليمة لضرب عددين بدون إشارة لتنتج عدد بدون إشارة أيضا . والشكل العام لهذه التعليمة هو كما يلي :

**والصيغة العامة لها :** MUL operand

**حيث أن operand تأخذ الحالات التالية :**

|  |  |
| --- | --- |
| MUL | REG memory |

**Algorithm**

when operand is a **byte**:  
 AX = AL \* operand.

when operand is a **word**:

( DX AX) = AX \* operand.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mnemonic** | **Meaning** | **Format** | **Operation** | **Flag effected** |
| MUL | unsigned Multiply | MUL S | AL **.** S8 → AX  AL **.** S16 → DX , AX | أعلام الحالة |

**ملاحظات** :

**1**-  **ايعازات الضرب تطبق على السجل AL فقط**

**2- اذا اردنا ضرب عددين فاحدهما يخزن في AL والأخر في أي سجل نريده .**

**3- اذا الضرب بعدد byte سوف توضع النتيجة بـ AX**

**4- اما اذا كان الضرب بعدد word سوف توضع النيجة بـ Dx , AX**

**5- اعلام الحالة تتاثر تبعا لنتيجة تنفيذ التعليمة**

**مثال 1:**

**Org 100h**

**MOV AL, 200 ; AL = 0C8h**

**MOV BL, 4**

**MUL BL ; AX = 0320h (800)**

**RET**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| r | ? | ? | r | ? | ? |

CF=OF=0 when high section of the result is zero.

**مثال2 :**

**org 100h**

**MOV AL, 3 ; AL = 03h**

**MOV BL, 4 ; bL = 04h**

**MUL BL ; AX = 000Ch (12)**

**RET**

**مثال 3:**

**org 100h**

**MOV Ax ,300 ; Ax = 012ch**

**MOV bx ,300 ; Bx = 012ch**

**MUL Bx ; AX = 5f90h**

**RET**

**نتيجة خاصل الضرب 90000 بالعشري ، أي 15f90 بالسادسي عشر ، لذا النتيجة الكلمة الادنى في Ax=5f90 والكلمة العليا في DX=0001**

**2- تعليمة Integer Multiply \_ IMUL**

تستعمل هذه التعليمة لضرب الأعداد بإشارة .

**والصيغة العامة لهذه التعليمة :**

**IMUL operand**

**حيث أن operand تأخذ الحالات التالية :**

|  |  |
| --- | --- |
| IMUL | REG memory |

**Signed multiply.**   
  
Algorithm:

when operand is a **byte**:  
AX = AL \* operand.

when operand is a **word**:  
(DX AX) = AX \* operand.

حيث تعمل على ضرب قيمتين الأولى منها إما ان تكون مخزنة في في مسجل او موقع في الذاكرة طوله بايت او كلمة والمعامل الثاني اما أن يكون المسجل AL او AX وذلك بناء على طول المعامل الأول ، ومن الجدير بالذكر أنه من غير الصواب ضرب قيمة فورية مباشرة ، فالعملية IMUL 10 هي عملية غير صحيحة ، ولإنجاز مثل هذه العملية يجب تخزين القيمة في مسجل او موقع في الذاكرة أولا ومن ثم يتم إنجاز العملية وكما يلي :

MOV BL , 10

IMUL BL

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mnemonic** | **Meaning** | **Format** | **Operation** | **Flag effected** |
| IMUL | signed Multiply | IMUL S | AL **.** S8 → AX  AL **.** S16 → DX , AX | أعلام الحالة |

**مثال1 :**

**Org 100h**

**MOV AL, -2**

**MOV BL, -4**

**IMUL BL ; AX = 8**

RET

**مثال2 :**

**org 100h**

**MOV Ax ,-300 ; Ax = 0FED4h**

**MOV bx ,-300 ; Bx = 0FED4h**

**IMUL Bx ; AX = 5f90h**

**RET**

**مثال 3:**

**org 100h**

**MOV Ax ,-300 ; Ax = 0FED4h**

**MOV bx ,-300 ; Bx = 0012Ch**

**IMUL Bx ; AX = 0A070h**

**RET**

**ان نتيجة الضرب تكون -90000 بالعشري أي -15f90 بالسداسي عشر وباخذ المتتم تكون النتيجة في المسجلين**

**AX= A070 و Dx=FFFE أي**

**3- تعليمة unsigned divide – DIV**

تستعمل هذه التعليمة مع الاعداد بدون اشارة والصيغة العامة لهذا الامر كالاتي :

DIV operand

حيث ان operand تاخذ الحالات التالية :

|  |  |
| --- | --- |
| DIV | REG memory |

**Unsigned divide**.   
  
**Algorithm:**

when operand is a **byte**:  
AL = AX / operand  
AH = remainder (modulus)

when operand is a **word**:  
AX = (DX AX) / operand  
DX = remainder (modulus)

وبالنسبة للأعلام الحالة تتأثر تبعا لنتيجة العملية :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| ? | ? | ? | ? | ? | ? |

**مثال1 :**

**org 100h**

**MOV Ax ,9 ; Ax = 09h**

**MOV BL ,3 ; Bx = 03h**

**DIV BL ; AX = 0003h**

**RET**

**مثال 2:**

**org 100h**

**MOV Ax ,9 ; Ax = 09h**

**MOV bl ,2 ; Bx = 03h**

**div bl ; AX = 0104h**

**RET**

**مثال 3 :**

**org 100h**

**MOV Ax ,9 ; Ax = 09h**

**MOV bx ,2 ; Bx = 03h**

**div bx ; AX = 0004h**

**RET**

**يحب ان نلاحظ انه في هذا المثال تم القسمة على معامل بطول 16 لذا ان باقي القسمة يكون في المسجل Dx=0001h**

**4 – تعليمة Integer Divide – IDIV**

عمل هذه التعليمة مشابه للتعليمة السابقة مع فرق رئيسي هو ان الأعداد هنا تحمل أشارة والصيغة العامة .

DIV operand

حيث أن operand تأخذ الحالات التالية :

|  |  |
| --- | --- |
| DIV | REG memory |

**Signed divide.**   
  
**Algorithm**:

when operand is a **byte**:  
AL = AX / operand  
AH = remainder (modulus)

when operand is a **word**:  
AX = (DX AX) / operand  
DX = remainder (modulus)

\* وبالنسبة للأعلام الحالة تتأثر تبعا لنتيجة العملية :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| ? | ? | ? | ? | ? | ? |

**مثال1:**

**org 100h**

**MOV Ax ,-9 ; Ax = 0FFF7h**

**MOV bL ,2 ; Bx = 03h**

**Idiv bL ; AX = 0FFFCh**

**RET**

**مثال2:**

**org 100h**

**mov ax , 9**

**mov bx , -3**

**idiv bx**

**ret**

**ناقش تنفيذ المثال الثاني؟**

**مثال : جد حل هذه المعادلة ، ؟**

**a2 + a + b – c + (a + b)2 / c**

**حيث ان أي قيم a , b , c >=1**

**Mov CL , a**

**ADD CL , b**

**Mov AL , a**

**MUL AL**

**Add AL , CL**

**Sub AL , C**

**Mov BL , AL**

**Mov AL , CL**

**MUL AL**

**Div C**

**Add BL , AL**

**2) لتكن a=2 , b=3, c=4, d=1 أكتب برنامج بلغة الاسمبلي لايجاد قيم التعابير الحسابية التالية**

1. **a + b \* c**
2. **a – (b +d ) + 10**
3. **a2 + b2**
4. **d + (c / 2 ) \* b2 + 5**
5. **y = 4a2 + 2b2 – c**

**3) الايعازات المنطقية Logic Instruction**

**1- تعليمات معالجة البيانات الثنائية Bit Manipulation Instructions**

سميت هذه التعليمات بهذا الاسم لكونها تعمل حسب قواعد المنطق الشكلي (Formal logic) وليس حسب القواعد الرياضية ، والجدول التالي يبين التعليمات المنطقية :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mnemonic** | **Meaning** | **Format** | **Operation** | **Flag effected** |
| AND | Logical And | AND D , S | D ← S . D | أعلام الحالة |
| OR | Logical OR | OR D , S | D ← S + D | أعلام الحالة |
| XOR | Exclusive OR | XOR D ,S | D ← S ○ D | أعلام الحالة |
| NOT | Logical not | Not D | D ← D | None |

**\* الإيعاز AND**

يأخذ الإيعاز الصيغة العامة التالية : And operand1 , opearnd2

حيث أن المعاملين operand1 , operand2 يأخذان الحالات التالية :

|  |  |
| --- | --- |
| **AND** | **REG, memory memory, REG REG, REG memory, immediate REG, immediate** |

هذا الإيعاز يطبق على معاملين وتطبق عليهما الخوارزمية الآتية :

1. يحول كلا المعاملين الى ما يطابقهما بشفرة الـ BCD ( أي الى 0 و 1 ) .
2. تطبق خوارزمية and كالاتي :

These rules apply:

1 AND 1 = 1

1 AND 0 = 0

0 AND 1 = 0

0 AND 0 = 0

1. الناتج يتم إرجاعه إلى صيغة السداسي عشر

**مثال توضيحي** :

**ليكن**  AL= 87 , BL= f1 ، And AL , BL

**ملاحظة :**  لا يجوز تطبيق هذه الايعازات على معاملات مختلفة بالطول .

AL=87 → 1000 0111

BL=F1 → 1111 0001

\_\_\_\_\_\_\_\_\_\_\_\_\_

AL 1000 0001 = 81

مثال :

Org 100h

Mov AL , 2

Mov BL , 10

And AL ,BL

Ret

مثال :

Org 100h

MOV AL, 'a' ; AL = 01100001b

AND AL, 11011111b ; AL = 01000001b ('A')

RET

**\* الإيعاز OR**

يأخذ الإيعاز الصيغة العامة التالية : OR operand1 , opearnd2

حيث أن المعاملين operand1 , operand2 يأخذان الحالات التالية :

|  |  |
| --- | --- |
| OR | REG, memory memory, REG REG, REG memory, immediate REG, immediate |

ان الايعاز OR يعمل بنفس خوارزمية and ما عدا أن OR تعمل حسب الجدول التالي :

These rules apply:

1 OR 1 = 1

1 OR 0 = 1

0 OR 1 = 1

0 OR 0 = 0

**مثال توضيحي** :

**ليكن**  AX= 8102 , CX= F12A ، OR AL , CH

**ملاحظة :**  لا يجوز تطبيق هذه الايعازات على معاملات مختلفة بالطول .

AL= 02 → 1000 0010

CH=F1 → 1111 0001

\_\_\_\_\_\_\_\_\_\_\_\_\_

AL 1111 0011 = F3

**مثال :**

**Org 100h**

**Mov ax, 08102h**

**Mov cx, 0f12ah**

**Or AL , CH**

**Ret**

**مثال :**

**Org 100h**

**MOV AL, 'A' ; AL = 01000001b**

**OR AL, 00100000b ; AL = 01100001b ('a')**

**RET**

**\* الإيعاز NOT**

يأخذ الإيعاز الصيغة العامة التالية : NOT operand1

حيث أن المعامل operand1 يأخذ الحالات التالية :

|  |  |
| --- | --- |
| NOT | REG memory |

هذا الإيعاز يطبق على معامل واحد فقط ويقوم بقلب البتات ، كالاتي :

**Invert each bit of the operand.  
Algorithm**:

* if bit is 1 turn it to 0.
* if bit is 0 turn it to 1.

وذلك بطرح FF من العدد اذا كان يتكون من Byte او من FFFF اذا كان يتكون من word

**مثال توضيحي :**

**لتكن** AL = 86 اذن not AL تطبق كالأتي:

Not AL → FF - 86 → AL= 79

**مثال :**

Org 100h

Mov AL , 11001100b

Mov BX, 01222h

Not AL

Not BX

Ret

**ملاحظة :**

**Not + 1 → Neg**

**Neg – 1 → Not**

**\* الإيعاز XOR**

يأخذ الإيعاز الصيغة العامة التالية : XOR operand1 , opearnd2

حيث أن المعاملين operand1 , operand2 يأخذان الحالات التالية :

|  |  |
| --- | --- |
| XOR | REG, memory memory, REG REG, REG memory, immediate REG, immediate |

يستخدم هذا الايعاز بتطبيقه على معاملين كالاتي :

**Logical XOR (Exclusive OR) between all bits of two operands. Result is stored in first operand.**  
These rules apply:

1 XOR 1 = 0

1 XOR 0 = 1

0 XOR 1 = 1

0 XOR 0 = 0

أي ان:

1. اذا كانت البتات متشابهة نضع القيمة 0
2. اذا كانت البتات مختلفة نضع لها القيمة 1

**مثال توضيحي :**

ليكن Ax= 6172 و Dx= A100 ، XOR AL , DH

AL = 72 → 0111 0010

DH = A1 → 1010 0001

\_\_\_\_\_\_\_\_\_\_\_\_

AL = 53 1101 0011

**ملاحظة : ومما يجب التنويه له أن كل تعليمة منطقية قد تستخدم لانجاز مهام معينة فمثلا** :

***Uses AND***

**1 – To check specified one / several bits**

Exp : And AL , 01 ; if ZF=1 then Al is even else If ZF=0 then AL is odd

**2- To reset any selected bits**

اذا اردنا تصفير بت معين فقط نجعل فيمة البت المقابل له في ايعاز and يساوي 0 والباقي تترك أي بوضع 1

**Exp1**: reset bit 3 of BL 11110111= f7

And BL , F7

**Exp2** : Reset bits 1,2,5 of AL

AND AL , D9

**Exp3:** reset bits 2,4,6,11 of BX

AND BX , F7AB

***Uses OR***

**Its used to set the selected bits ( made them 1 )**

**Exp1:** Set bit 4 of AL 0001 0000

OR AL , 10

**EXP2** : Set bit 1,2,5 of AL

OR AL , 26

***Uses XOR***

**1 – invert the selected bits**

**Exp1:** invert bits 1,3,5 of CL 0010 1010

XOR CL , 2A

**EXP2** : invert bits 3 ,5,6,7 of AL

XOR AL , E8

**2- It is used to complement all bit of any reg**

**Uses NOT**

**It is used to complement all bits of any Reg (1** → 0 , 0 → 1 **)**

**\* تعليمة Convert Byte to word – CBW**

تستعمل هذه التعليمة لتكرار خانة الإشارة MSB of AL في المسجل AL الى جميع الخانات الثنائية في المسجل AH ، وتعطي هذه التعليمة المبرمج القدرة على جمع عدد طوله بايت الى عدد طوله كلمة او طرح عدد طوله عدد كلمة من عدد آخر طوله بايت ، ويأخذ الصيغة العامة التالية : ويمكن ان نلاحظ ان التعليمة لاتاخذ أي معلمات .

|  |  |
| --- | --- |
| CBW | No operands |

ويمكن تلخيص عمل الايعاز بالتالي :

Convert byte into word.   
  
Algorithm:   
  
if high bit of AL = 1 then:

* AH = 255 (0FFh)

else

* AH = 0

Example:

Org 100h

MOV AX, 0 ; AH = 0, AL = 0

MOV AL, -5 ; AX = 000FBh (251)

CBW ; AX = 0FFFBh (-5)

RET

أما أعلام الحالة فلا تتأثر في هذه الإيعاز

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| unchanged | | | | | |

**\* تعليمة Convert Word to Double – CWD**

تستعمل هذه التعليمة لمد خانة الاشارة (MSB of AX) في المسجل AX الى جميع خانات المسجل DX . فباستعمال هذه التعليمة يستطيع المبرمج تقسيم معامل حسابي طوله كلمة على معامل حسابي اخر طوله كلمة ايضا . علما ان عملية القسمة تتطلب ان يكون طول العدد المقسوم ضعف طول المقسوم عليه ، والصيغة العامة لهذا الايعاز كالاتي :

|  |  |
| --- | --- |
| CWD | No operands |

ويمكن تلخيص عمل الايعاز بالتالي :

Convert Word to Double word.   
  
Algorithm:   
  
if high bit of AX = 1 then:

* DX = 65535 (0FFFFh)

else

* DX = 0

**والمثال التالي يوضح ذلك :**

Example:

Org 100h

MOV DX, 0 ; DX = 0

MOV AX, 0 ; AX = 0

MOV AX, -5 ; DX AX = 00000h:0FFFBh

CWD ; DX AX = 0FFFFh:0FFFBh

RET

أما أعلام الحالة فلا تتأثر في هذه الإيعاز :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| unchanged | | | | | |

1. **تعليمات الإزاحة shift instructions**

تعمل هذه التعليمات على ازاحة محتويات مسجل او مكان في الذاكرة طول كل منهما بايت او كلمة .ويوجد هنالك نوعان من الازاحة هما الازاحة المنطقية والازاحة الرياضية وكما هو موضح في الجدول التالي :

|  |  |  |  |
| --- | --- | --- | --- |
| **Mnemonic** | **Meaning** | **Format** | **Flag effected** |
| SAL | Shift arithmetic Left | SAL D , count | OF , CF |
| SHL | Shift Logical Left | SHL D, count | OF , CF |
| SHR | Shift Logical right | SHR D, count | أعلام الحالة |
| SAR | Shift arithmetic Right | SAR D, count | أعلام الحالة |

**ملاحظات :**

1. يتم في هذه الايعازات تزحيف البنات إلى الجهة المحددة عدد من المرات
2. اذا كان التزحيف بمرتبة واحدة فيكتب الايعاز SHL AL , 1
3. اذا كان التزحيف باكثر من مرتبه فيكتب الايعاز

Mov CL , 3 → عدد المرات الملراد تزحيفها

SHL AL , CL

1. يتم تعويض المراتب الفارغة كالاتي :
2. في الايعازات SHL ، SHR ، SAL بأصفار
3. في الايعاز SAR يتم تكرار البت الاخير (بت الاشارة) .

5- ان المعاملات المسموحة بالنسبة لتعليمات الإزاحة هي :

|  |  |
| --- | --- |
| **Count** | **D** |
| **1** | **Reg** |
| **CL** | **Reg** |
| **1** | **Mem** |
| **CL** | **Mem** |

أي عندما Countلا يساوي الواحد فعندئذ يجب تحويل قيمة count في المسجل CL ثم كتابة تعليمات الازاحة أي عندما Count يساوي الواحد ، فيمكن ان نكتب : SAL AX , 1 وعندما Count <> 1 يجب ان نكتب :

MOV CL , count

SAL AX , CL

وألان سنتعرف على كل إيعاز بالتفصيل :

* + **الايعاز SAL**

هذه التعليمة على ازاحة الاعداد ذات الاشارة الى اليسار . ولهذا فأن هذه التعليمة لا تحافظ على اشارة العدد فمن الممكن ان يصبح العدد الموجب سالبا وان يصبح العدد السالب موجبا بعد تنفيذ هذه التعليمة .والصيغة العامة لهذه التعليمة SAL opearnd1 , operand2 حيث ان :

|  |  |
| --- | --- |
| SAL | memory, immediate REG, immediate  memory, CL REG, CL |

Shift Arithmetic operand1 Left. The number of shifts is set by operand2.   
  
Algorithm:

* Shift all bits left, the bit that goes off is set to CF.
* Zero bit is inserted to the right-most position.

والمثال التالي يوضح عمل هذا الإيعاز :

Org 100h

MOV AL, 0E0h ; AL = 11100000b

SAL AL, 1 ; AL = 11000000b, CF=1.

RET

اما اعلام الحالة فتتأثر بهذا الايعاز كالأتي :

|  |  |
| --- | --- |
| C | O |
| r | r |

OF=0 if first operand keeps original sign.

**- الايعاز SHL**

تستعمل هذه التعليمة للأعداد بدون اشارة حيث يتم ازاحة الأعداد لليسار وان عمل هذه التعليمة مشابهه الى عمل التعليمة SAL ، والصيغة العامة SHL operand1 , opearnd2 ، حيث تأخذ المعلمات الحالات التالية :

|  |  |
| --- | --- |
| SHL | memory, immediate REG, immediate  memory, CL REG, CL |

ويمكن تلخيص عمل التعليمة التالية :

Shift operand1 Left. The number of shifts is set by operand2.   
  
Algorithm:

* Shift all bits left, the bit that goes off is set to CF.
* Zero bit is inserted to the right-most position.

والمثال التالي يوضح عمل هذا الايعاز

Example:

MOV AL, 11100000b

SHL AL, 1 ; AL = 11000000b, CF=1.

RET

اما اعلام الحالة التي تتاثر بهذا الايعاز

|  |  |
| --- | --- |
| C | O |
| r | r |

OF=0 if first operand keeps original sign.

* + **الإيعاز SAR**

تستعمل هذه التعليمة لإزاحة الأعداد ذات الاشارة الى اليمين ، اما بالنسبة للخانات الثنائية التي يتم تفريغها فيتم ازاحة اشارة العدد اليها ، والصيغة العامة لهذا الايعاز SAR Opearnd1 , operand2 ، حيث ان المعاملات تاخذ الحالات التالية :

|  |  |
| --- | --- |
| SAR | memory, immediate REG, immediate  memory, CL REG, CL |

**ويمكن تلخيص عمل الايعاز كالتالي :**

Shift Arithmetic operand1 Right. The number of shifts is set by operand2.   
  
Algorithm:

* Shift all bits right, the bit that goes off is set to CF.
* The sign bit that is inserted to the left-most position has the same value as before shift.

**والمثال التالي يوضح عمل الايعاز :**

Example:

Org 100h

MOV AL, 0E0h ; AL = 11100000b

SAR AL, 1 ; AL = 11110000b, CF=0.

MOV BL, 4Ch ; BL = 01001100b

SAR BL, 1 ; BL = 00100110b, CF=0.

RET

ما اعلام الحالة التي تتاثر بهذه العملية :

|  |  |
| --- | --- |
| C | O |
| r | r |

OF=0 if first operand keeps original sign.

* + **الايعاز SHR**

فتستعمل هذه التعليمة لإزاحة الأعداد الممثلة بدون أشارة ، وعند تنفيذ هذه التعليمة فان الخانة MSB المفرغة يتم خزن فيها القيمة 0 . والصيغة العامة لهذا الايعاز SAR Opearnd1 , operand2 ، حيث ان المعاملات تاخذ الحالات التالية :

|  |  |
| --- | --- |
| SAR | memory, immediate REG, immediate  memory, CL REG, CL |

ويمكن تلخيص عمل الايعاز كالاتي :

Shift operand1 Right. The number of shifts is set by operand2.   
  
Algorithm:

* Shift all bits right, the bit that goes off is set to CF.
* Zero bit is inserted to the left-most position.

والمثال التالي يوضح عمل الايعاز

Example:

Org 100h

MOV AL, 00000111b

SHR AL, 1 ; AL = 00000011b, CF=1.

RET

إما إعلام الحالة التي تتأثر بهذا الإيعاز :

|  |  |
| --- | --- |
| C | O |
| r | r |

OF=0 if first operand keeps original sign.

**ملاحظة :**

ويمكن استعمال عمليات الازاحة لانجاز عمليات الضرب والقسمة وبسرعة اكثر من استخدام تعليمات الضرب والقسمة مباشرة :

- حيث ان الازاحة الى اليسار بمقدار ثنائية واحدة تعني عملية الضرب × 2

- وان الازاحة الى اليمين موضعا واحدا تعادل القسمة على 2

مثال : ضرب قيمة المسجل AX بـ 5

Org 100h

MOV BX , AX

SHL AX ,1 ; Multiply AX \* 2

SHL AX ,1 ; Multiply AX \* 2

ADD AX , BX ; the final result in AX

H.w : find AX \* 10 ?

Solution:

shl ax, 1 ;Multiply AX by two

mov bx, ax ;Save 2\*AX for later

shl ax, 1 ;Multiply AX by four

shl ax, 1 ;Multiply AX by eight

add ax, bx ;Add in 2\*AX to get 10\*AX

**4) تعليمات التدوير للقيم الثنائية Rotate Instructions**

تشبه هذه التعليمات تعليمات الازاحة مع الفارق الرئيسي وهو ان تعليمات الدوران تحفظ القيم الثنائية المفرغة نتيجة الازاحة وذلك بأعادتها ثانية الى المعامل ومن الجهة الاخرى . وكما هو الحال في تعليمات الازاحة ، فأن القيم الثنائية المزاحة يتم تخزينها في راية الحمل CF أيضا . والجدول التالي يبين الصيغة العامة لها :

|  |  |  |  |
| --- | --- | --- | --- |
| **Mnemonic** | **Meaning** | **Format** | **Flag effected** |
| ROL | Rotate Left | ROL D , count | OF , CF |
| ROR | Rotate Right | ROR D ,count | OF , CF |
| RCL | Rotate Left through carry | RCL D, count | أعلام الحالة |
| RCR | Rotate Right through carry | RCR D, count | أعلام الحالة |

والجدول التالي يبين المعاملات المسموح بها للتعليمات التدوير :

|  |  |
| --- | --- |
| **Count** | **D** |
| **1** | **Reg** |
| **CL** | **Reg** |
| **1** | **Mem** |
| **CL** | **Mem** |

**وفيما يلي توضيح موجز لتلك التعليمات :-**

**- تعليمة ROL :** في هذه التعليمة نلاحظ أن القيمة الثنائية في الموقع MSB يتم انتقالها الى الموقع الأقل أهمية بالإضافة إلى راية الحمل CF ، والصيغة العامة لهذا الإيعاز كالتالي ROL operand1 , operand2 حيث ان :

|  |  |
| --- | --- |
| ROL | memory, immediate REG, immediate  memory, CL REG, CL |

Rotate operand1 left. The number of rotates is set by operand2.   
  
**Algorithm**: shift all bits left, the bit that goes off is set to CF and the same bit is inserted to the right-most position.

**Example:**

**Org 100h**

MOV AL, 1Ch ; AL = 00011100b

ROL AL, 1 ; AL = 00111000b, CF=0.

RET

إما إعلام الحالة التي تتغير عند تنفيذ هذه التعليمة كالتالي :

|  |  |
| --- | --- |
| C | O |
| r | r |

OF=0 if first operand keeps original sign.

**EXP** : IF AL =10110100 , and CF=1 , **then**  ROL AL ,1 **⇨** AL= 01101001 , CF=1

**- تعليمة ROR :** في هذه التعليمة نلاحظ ان القيمة المخزنة في الموقع LSB تنتقل الى الموقع MSB بالاضافة الى تخزينها ايضا في راية الحمل CF والصيغة العامة لهذه التعليمة ROR operand1 , operand2 كالتالي :

|  |  |
| --- | --- |
| ROR | memory, immediate REG, immediate  memory, CL REG, CL |

Rotate operand1 right. The number of rotates is set by operand2.   
  
**Algorithm**: shift all bits right, the bit that goes off is set to CF and the same bit is inserted to the left-most position.

**Example**:

Org 100h

MOV AL, 1Ch ; AL = 00011100b

ROR AL, 1 ; AL = 00001110b, CF=0.

RET

|  |  |
| --- | --- |
| C | O |
| r | r |

OF=0 if first operand keeps original sign.

**EXP** : IF AL =10110100 , and CF=1 , **then**  ROR AL ,1 **⇨** AL= 01011010 , CF=1

**- تعليمة RCR :**  في هذه التعليمة نلاحظ انه القيمة في الموقع LSB يتم نقلها الى راية الحمل CF وان محتويات راية الحمل قبل تنفيذ التعليمة يتم نقلها الى الموق MSB والصيغة العامة كالتالي : RCR operand , operand2

|  |  |
| --- | --- |
| RCR | memory, immediate REG, immediate  memory, CL REG, CL |

Rotate operand1 right through Carry Flag. The number of rotates is set by operand2.   
  
**Algorithm**: shift all bits right, the bit that goes off is set to CF and previous value of CF is inserted to the left-most position.   
**Example:**

Org 100h

STC ; set carry (CF=1).

MOV AL, 1Ch ; AL = 00011100b

RCR AL, 1 ; AL = 10001110b, CF=0.

RET

|  |  |
| --- | --- |
| C | O |
| r | r |

OF=0 if first operand keeps original sign.

**EXP** : IF AL =10110100 , and CF=1 , **then**  ROR AL ,1 **⇨** AL= 11011010 , CF=0

**- تعليمة RCL :** في هذه التعليمة كسابقتها يتم نقل القيمة الثنائية في راية الحمل الى الموقع LSB وقيمة الثنائية في الموقع MSB يتم نقلها الى راية الحمل CF والصيغة العامة لها RCL Opearnd1 , operand2 حيث ان :

|  |  |
| --- | --- |
| RCL | memory, immediate REG, immediate  memory, CL REG, CL |

Rotate operand1 left through Carry Flag. The number of rotates is set by operand2.   
When **immediate** is greater then 1, assembler generates several **RCL xx, 1** instructions because 8086 has machine code only for this instruction (the same principle works for all other shift/rotate instructions).   
  
**Algorithm**: shift all bits left, the bit that goes off is set to CF and previous value of CF is inserted to the right-most position.   
**Example:**

Org 100h

STC ; set carry (CF=1).

MOV AL, 1Ch ; AL = 00011100b

RCL AL, 1 ; AL = 00111001b, CF=0.

RET

|  |  |
| --- | --- |
| C | O |
| r | r |

OF=0 if first operand keeps original sign.

**EXP** : IF AL =10110100 , and CF=1 , **then**  ROL AL ,1 **⇨** AL= 01101001 , CF=1

**5) تعليمات السيطرة على الرايات Flags Control Instruction**

وتتضمن مجموعة من التعليمات التي يتم من خلالها تحديث او تغيير قيم الرايات والجدول التالي يبين أهمها :

|  |  |  |
| --- | --- | --- |
| **Operation** | **Meaning** | **Mnemonic** |
| CF = 0 | Clear carry flag | CLC |
| CF = 1 | Set carry flag | STC |
| CF= not CF | Complement carry flag | CMC |

**6) المتغيرات Variables**

Variable is a memory location. For a programmer it is much easier to have some value be kept in a variable named "**var1**" then at the address 5A73:235B, especially when you have 10 or more variables.   
  
Our compiler supports two types of variables: **BYTE** and **WORD**.

|  |
| --- |
| Syntax for a variable declaration:  *name* **DB** *value*  *name* **DW** *value*  **DB** - stays for Define Byte. **DW** - stays for Define Word.  *name* - can be any letter or digit combination, though it should start with a letter. It's possible to declare unnamed variables by not specifying the name (this variable will have an address but no name).  *value* - can be any numeric value in any supported numbering system (hexadecimal, binary, or decimal), or "**?**" symbol for variables that are not initialized  **Example :**  ORG 100h  MOV AL, var1  MOV BX, var2  RET ; stops the program.  VAR1 DB 7  var2 DW 1234h |

**6) تعليمات المكدس stack instructions**

**1- الإيعاز pop :** في هذا الإيعازيتم سحب كلمة من قمة المكدس والصيغة العامة لهذا الإيعاز كالتالي POP Operand1 ، حيث أن :

|  |  |
| --- | --- |
| **POP** | **REG SREG memory** |

ويمكن تلخيص عمل هذا الإيعاز بالشكل التالي :

Get 16 bit value from the stack.   
  
Algorithm:

* operand = SS:[SP] (top of the stack)
* SP = SP + 2

Example:

Org 100h

MOV AX, 1234h

PUSH AX

POP DX ; DX = 1234h

RET

أما أعلام الحالة لا تتأثر عند تنفيذ هذا الايعا

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| unchanged | | | | | |

**2- الإيعاز Push :** في هذا الإيعازيتم إضافة كلمة من قمة المكدس والصيغة العامة لهذا الإيعاز كالتالي Push Operand1 ، حيث أن :

|  |  |
| --- | --- |
| PUSH | REG SREG memory immediate |

ويمكن تلخيص عمل هذا الإيعاز بالشكل التالي :

Store 16 bit value in the stack.  
  
Note: **PUSH immediate** works only on 80186 CPU and later!   
  
Algorithm:

* SP = SP - 2
* SS:[SP] (top of the stack) = operand

**Example:**

Org 100h

MOV AX, 1234h

PUSH AX

POP DX ; DX = 1234h

RET

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| unchanged | | | | | |

**3- الإيعاز Call :** هذا الإيعازيستخدم للانتقال عند استدعاء نهج معالجة او الذهاب الى عنوان معين والصيغة العامة لهذا الإيعاز كالتالي:

|  |  |
| --- | --- |
| CALL | procedure name label 4-byte address |

ويمكن تلخيص عمل هذا الإيعاز بالشكل التالي :

Transfers control to procedure, return address is (IP) is pushed to stack. *4-byte address* may be entered in this form: 1234h:5678h, first value is a segment second value is an offset (this is a far call, so CS is also pushed to stack).  
  
  
Example:

ORG 100h ; for COM file.

CALL p1

ADD AX, 1

RET ; return to OS.

p1 PROC ; procedure declaration.

MOV AX, 1234h

RET ; return to caller.

p1 ENDP

وأعلام الحالة لا تتأثر بهذا الإيعاز :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| unchanged | | | | | |

**4- الإيعاز Ret :** هذا الإيعاز يستخدمعند الانتهاء نهاية البرنامج او عند نهاية الانهجة الفرعية والصيغة العامة لهذا الإيعاز كالتالي :

|  |  |
| --- | --- |
| RET | No operands or even immediate |

ويمكن تلخيص عمل هذا الإيعاز بالشكل التالي :

Return from near procedure.   
  
**Algorithm**:

* Pop from stack:
  + IP
* if immediate operand is present: SP = SP + operand

**Example:**

ORG 100h ; for COM file.

CALL p1

ADD AX, 1

RET ; return to OS.

p1 PROC ; procedure declaration.

MOV AX, 1234h

RET ; return to caller.

p1 ENDP

وأعلام الحالة لا تتأثر بهذا الإيعاز :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| unchanged | | | | | |

**7) تعليمة المقارنة Compare instruction CMP**

تسمح تعليمة المقارنة CMP بمقارنة عددين ب 8 بت او 16 بت والصيغة العامة كالأتي :

CMP operand1 , operand2 ، تجري عملية الطرح ضمنيا دون تخزين نتيجتها في معامل الهدف ( أي تبقى كلا من محتويات المصدر ومحتويات الهدف على حالها ) وتستعمل هذه التعليمة لجعل أعلام الحالة تأخذ قيمة (0 او 1) ،وان المعاملات المسموح بها لهذه التعليمة مبينة في الجدول التالي :

|  |  |
| --- | --- |
| CMP | REG, memory memory, REG REG, REG memory, immediate REG, immediate |

ويمكن تلخيص عمل هذا الإيعاز بالشكل التالي :

**Compare.**   
  
**Algorithm:**  
  
operand1 - operand2   
  
result is not stored anywhere, flags are set (OF, SF, ZF, AF, PF, CF) according to result.

**Example:**

**Org 100h**

MOV AL, 5

MOV BL, 5

CMP AL, BL ; AL = 5, ZF = 1 (so equal!)

RET

وأعلام الحالة تتأثر تبعاً لنتيجة تنفيذ هذا الإيعاز

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| r | r | r | r | r | r |

**8) تعليمات نقل التحكم Control Transfer Instructions**

تعمل هذه التعليمات على نقل سير تنفيذ البرنامج من مكان إلى أخر ، ويمكن تقسيم هذه التعليمات إلى أربعة مجموعات رئيسية وكما يلي :

**تعليمة نقل التحكم غير المشروط Un conditional Jump - JMP**

**تعليمات نقل التحكم المشروط Conditional Transfer instructions**

**تعليمات التكرار Iteration control instructions**

**تعليمات استدعاء البرنامج الفرعي والرجوع منه Procedure Call And Return**

وتجدر الإشارة الى ان هذه التعليمات لا تؤثر على الرايات flags . أما بالنسبة لعناوين المواقع التي يتم انتقال التحكم اليها فتقسم الى ثلاث انواع وذلك حسب التعليمة وموقع المكان الذي سيتم الانتقال إليه وكما يلي

:

1) عناوين قصيرة Short Labels : التي تستعمل عادة مع تعليمات التكرار وتعليمات التحكم المشروط .

2) عناوين قريبة Near Labels : فتستعمل عادة مع تعليمة استدعاء البرنامج الفرعي ومع تعليمة النقل الغير مشروط التي من غير ممكن ان تكون قصيرة .

3) عناوين البعيدة Far labels : وهي تلك العناوين التي تؤثر على مسجل قطاع البرنامج CS .

وفيما يلي استعراض موجز لعمليات نقل التحكم :

**أ) الإيعاز JMP( نقل التحكم الغير مشرط ) :** تشبه هذه التعليمة تعليمة GOTO في لغتي بيسك وباسكال ، فعندما ينفذ المعالج تعليمة JMP ينتقل تنفيذ البرنامج الى العنوان المحدد في التعليمة والصيغة العامة لهذه التعليمة هي : JMP target ، حيث ان target هو الهدف او عنوان التعليمة المراد الانتقال اليها ، مع ملاحظة أن تعليمة JMP تغير قيمة محتوى سجل IP بقيمة جديدة وهي Target . والصيغة العامة لهذا الإيعاز كالأتي :

|  |  |
| --- | --- |
| JMP | label 4-byte address |

ويمكن تلخيص عمل هذا الإيعاز بالشكل التالي :

**Unconditional Jump**. Transfers control to another part of the program. *4-byte address* may be entered in this form: 1234h:5678h, first value is a segment second value is an offset.  
  
  
**Algorithm:**

Always jump

**Example:**

ORG 100h

MOV AL, 5

JMP label1 ; jump over 2 lines!

MOV AL, 0

label1:

Mov BL ,6

RET

وأعلام الحالة لا تتأثر بهذا الإيعاز

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| unchanged | | | | | |

**ب) تعليمات نقل التحكم المشروط Conditional Transfer instructions :**

أن الصيغة العامة لعملية لنقل مشروط المبينة في الجدول التالي :JCC Label

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mnemonic** | **Meaning** | **Format** | **Operation** | **Flag effected** |
| Jcc | Conditional Jemp | Jcc operand | اذا تحقق الشرط CC فانه يتم القفز الى العنوان المحدد في المعامل operand والا يتم الانتقال الى التعليمة التالية لتعليمة القفز | None |

**\* CC تمثل الشرط الذي سيتم الانتقال تبعاً لتحققه لذل سيستبدل بواحدة عند كل ايعاز حسب الشرط ، و توفر لغة الاسمبلي أكثر من 31 تعليمة سنبين أهمها في الجدول التالي :**

|  |  |
| --- | --- |
| **Meaning** | **Mnemonic** |
| If CF=1 | JC |
| If CF=0 | JNC |
| If OF=1 | JO |
| If OF=0 | JNO |
| If SF=1 | JS |
| If SF=0 | JNS |
| If CX=0000 | JCXZ |
| If equal / the result=0 | JE / JZ |
| If not equal / not less | JGE / JNL |
| If above /if not below nor equal cf=0 and zf=0 | JA / JNBL |
| If above or equal / if not below cf=0 | JAE / JNB |
| If below / if not above , cf=1 | JB / JNAE |
| If below or equal / if not above CF=1 and zf=1 | JBE / JNA |
| If greater/ if not less nor equal zf=0 and sf=0 | JG /JNLE |
| If less or equal / if not grater | JLE / JNG |
| If not equal / if not zero zf=0 | JNE / JNZ |
| If not parity (odd)/ if parity odd Pf=0 | JNP / JPO |
| If no parity (even) / if parity even ,PF=1 | JP / JPE |

**ملاحظة :**  للتمييز بين مقارنة الأعداد ذات الاشارة والاعداد بدون اشارة فان هنالك اسمين مختلفين يبدو انهما نفس الشيء في تعليمات القفز وهما فوق (A) وتحت (B) من اجل مقارنة الاعداد بدون اشارة ، واصغر L واكبر G من اجل مقارنة الاعداد ذات الاشارة .

**وفيما يلي بعض الامثلة عن استعمال تعليمات نقل التحكم المشروط :**

1- ADD AL , BL 2 – SUB AL , BL 3- CMP AL , BL 4 – CMP BX , AX

JC LOC JZ ZERO JE label1 JA LOC1

والجدول التالي يلخص انواع جمل نقل التحكم المشروط والواجب استعمالها بعد تعليمة CMP وذلك لتغطية الحالات المختلفة بين معامل الوهة ومعامل المصدر في تعليمة CMP .

|  |  |
| --- | --- |
| في حالة الاعداد باشارة | في حالة الاعداد بدون اشارة |
| JG | JA |
| JE | JE |
| JL | JB |
| JNE | JNE |
| JLE | JBE |
| JGE | JAE |

فيمكن أن نستخدم خليط من تعليمة CMP وتعليمات النقل المشروط للمحاكاة بعض التعليمات في اللغات العلية مثل If cond then stat او if cond then stat1 else stat2 او for stat او while cond stat او repeat . مع الملاحظة أنه في حالة الانتقال الى الاسفل يعكس الشرط ، اما اذا كان الانتقال الى الاعلى فيبقى الشرط كما هو .

**Exp1**  : **to simulate if cond then stat1**

If c > 20 the stat1

CMP C , 20

JLE REST

……

stat1

rest :

بقية البرنامج

**Exp2** : **If cond then stat1 Else Stat2**

If a > b then stat1 else stat2

CMP a , b

JG then

Else :

……

stat2

…….

Jmp rest

Then :

……

stat1

……

rest :

……..

بقية البرنامج

**EXP3 :**  **For i=N1 To N2 Step N3 stat**

MOV AL , N1

Rep:

stat

INC AL

OR

ADD AL ,N3

CMP AL , N3

JLE REP

بقية البرنامج

اذا كانت الزيادة بواحد نستخدم INC

اما اذا كانت اكثر من واحد نستخدم add

**EXP4** : **While Cond Do**

**…….**

**Stat**

Rep : cmp v1 ,v2

J ( عكس الشرط) Exit

…….

Stat

……

jmp rep

Exit :

**بقية البرنامج**

**ت ) تعليمات التكرار Iteration Instructions**

توفر لغة الاسمبلي العدد من تعليمات التكرار سنبين في هذه الفقرة أهمها ، وتستعمل هذه التعليمات المسجل CX لتخزين عدد مرات تنفيذ مجموعة جمل لغة التجميع وكل مرة يتم فيها تنفيذ هذه المجموعة من الجمل تسمى دورة . وفي كل دورة وعند تنفيذ تعليمة التكرار فانه يتم فيها انقاص قيمة المسجل CX بمقدار ومن ثم يتم اتخاذ القرار بواسطة المعالج بنقل التحكم الى بداية الدورة او عدمة وذلك بناء على القيمة المخزنة في المسجل CX . فاذا كانت القيمة المخزنة في CX لا تساوي 0 فانه يتم نقل التحكم الى بداية الدورة والا فانه يتم الخروج من مجموعة الجمل الى الجملة التالية لتعليمة التكرار ، يجب ملاحظة ان تخزين القيمة البدائية الصحيحة في المسجل CX تقع على عاتق المبرمج ويجب تخزين هذه القيمة قبل الجمل المراد تنفيذها عدة مرات .

**1 - تعليمة LOOP :**  والشكل العام لهذه التعليمة كما يلي : Location LOOP

Exp :

MOV CX , 10

XX : ………

……… True

……… If CX <> 0

LOOP XX

🠛 FALSE

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LOOP | label | Decrease CX, jump to label if CX not zero.   Algorithm:   * CX = CX - 1 * if CX <> 0 then   + jump   else   * + no jump, continue   Example:  include 'emu8086.inc'  ORG 100h  MOV CX, 5  label1:  PRINTN 'loop!'  LOOP label1  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | unchanged | | | | | | |

**\* الإيعاز LEA**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LEA | REG, memory | Load Effective Address.   Algorithm:   * REG = address of memory (offset)   Example:  MOV BX, 35h  MOV DI, 12h  LEA SI, [BX+DI] ; SI = 35h + 12h = 47h  Note: The integrated 8086 assembler automatically replaces **LEA** with a more efficient **MOV** where possible. For example:  org 100h  LEA AX, m ; AX = offset of m  RET  m dw 1234h  END   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | unchanged | | | | | | |

**مجموعة من الامثلة : عن if condition the stat**

org 100h

Mov al , 30

Mov bl ,20

Cmp al , bl

JLE exit

Add al ,bl

Jmp end

exit:

Ret

include 'emu8086.inc'

ORG 100h

MOV AL, 00000111b ; AL = 7

OR AL, 0 ; just set flags.

JNZ label1

PRINT 'zero.'

JMP exit

label1:

PRINT 'not zero.'

exit:

RET

**Q1: If the contents of the stack is shown in the Figure, trace the following instructions, the find the content of the specified registers shown in front of each instruction:**

|  |  |  |
| --- | --- | --- |
| **2008** | **77** | **A4** |
| **2006** | **B7** | **F2** |
| **2004** | **02** | **03** |
| **2002** | **12** | **34** |
| **2000** | **56** | **78** |

**Initial Stack**

**SP**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Instructions** | | **AX** | **BX** | **CX** | **SP** |
|  | | **- - -** | **- - -** | **- - -** | **2004** |
| **POP** | **CX** |  |  | **0302** | **2006** |
| **MOV** | **BX , 2E3C** |  | **2E3C** |  |  |
| **POP** | **AX** | **F2B7** |  |  | **2008** |
| **XCHG** | **AH , BL** | **3CB7** | **2EF2** |  |  |
| **SHR** | **AL , CL** | **3C2D** |  |  | |  |  |  | | --- | --- | --- | | **2008** | **77** | **A4** | | **2006** | **1E** | **2D** | | **2004** | **02** | **03** | | **2002** | **12** | **34** | | **2000** | **56** | **78** |   FINAL STACK |
| **NEG** | **BH** |  | **D2F2** |  | **SP** |
| **ROL** | **AH , CL** | **F02D** |  |  |  |
| **NOT** | **BL** |  | **D2OD** |  |  |
| **SUB** | **AX , BX** | **1E20** |  |  |  |
| **ADD** | **AL , BL** | **1E2D** |  |  |  |
| **PUSH** | **AX** |  |  |  | **2006** |

**Q2: If the contents of the stack is shown in the Figure, trace the following instructions, the find the content of the specified registers shown in front of each instruction:**

|  |  |  |
| --- | --- | --- |
| **0159** | **31** | **31** |
| **0157** | **99** | **23** |
| **0155** | **21** | **11** |
| **0153** | **34** | **12** |
| **0151A** | **-** | **-** |

**Initial Stack**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Instructions** | | **AX** | **BX** | **CX** | **DX** | **SP** |
|  | | **- - -** | **- - -** | **- - -** | **- - -** | **0153** |
| **MOV** | **DX , 0219Ah** |  |  |  | **219A** | **SP** |
| **Push** | **DX** |  |  |  |  | **0151** |
| **POP** | **AX** | **219A** |  |  |  | **0153** |
| **POP** | **BX** |  | **1234** |  |  | **0155** |
| **ADD** | **AL , BH** | **21AC** |  |  |  | |  |  |  | | --- | --- | --- | | **0159** | **31** | **31** | | **0157** | **99** | **23** | | **0155** | **21** | **11** | | **0153** | **34** | **12** | | **0151** | **9A** | **21** |   **Final Stack** |
| **POP** | **CX** |  |  | **1121** |  | **0157**  **SP** |
| **SUB** | **BL , CH** |  | **1223** |  |  |  |
| **AND** | **BX , CX** |  | **1021** |  |  |  |
| **ADD** | **BH , AL** |  | **BC21** |  |  |  |
| **SHL** | **AX , 1** | **4358** |  |  |  |  |

**Q3: If the contents of the stack is shown in the Figure, trace the following instructions, the find the content of the specified registers shown in front of each instruction : WHERE CF=0**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Instructions** | | **CL** | **BL** | **AL** |
| **MOV** | **CL , 2F** |  |  |  |
| **ROL** | **CL , 1** |  |  |  |
| **ROR** | **AL , 1** |  |  |  |
| **ADD** | **AL , 3** |  |  |  |
| **RCR** | **CL , 1** |  |  |  |
| **SUB** | **AL , 2** |  |  |  |
| **MOV** | **BL , AL** |  |  |  |
| **RCL** | **BL , 1** |  |  |  |
| **SHL** | **CL ,1** |  |  |  |
| **SHR** | **BL , 1** |  |  |  |

**Q4: If the contents of the stack is shown in the Figure, trace the following instructions, the find the content of the specified registers shown in front of each instruction:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Instructions** | | **AX** | **BX** | **CX** | **SP** |
|  | | **- - -** | **- - -** | **- - -** | **0202** |
| **MOV** | **CL , 3** |  |  |  | |  |  |  | | --- | --- | --- | | **0208** | **22** | **44** | | **0206** | **33** | **11** | | **0204** | **12** | **23** | | **0202** | **11** | **22** | | **0200** |  |  |   **Initial Stack** |
| **POP** | **AX** |  |  |  |  |
| **POP** | **BX** |  |  |  |  |
| **SHL** | **AL , 1** |  |  |  | **SP** |
| **ROR** | **BL , CL** |  |  |  |  |
| **XCHG** | **AL , AH** |  |  |  |  |
| **PUSH** | **AX** |  |  |  |  |
| **POP** | **CX** |  |  |  |  |